Privacy-Preserving Activity Scheduling on Mobile Devices

Igor Bilogrevic Murtuza Jadliwala Jean-Pierre Hubaux
Laboratory for Computer communications and Applications (LCA 1)
EPFL, CH-1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

Imad Aad Valtteri Niemi
Nokia Research Center
CH-1015 Lausanne, Switzerland
firstname.lastname@nokia.com

ABSTRACT

Progress in mobile wireless technology has resulted in the increased use of mobile devices to store and manage users' personal schedules. Users also access popular context-based services, typically provided by third-party providers, by using these devices for social networking, dating and activity-partner searching applications. Very often, these applications need to determine common availabilities among a set of user schedules. The privacy of the scheduling operation is paramount to the success of such applications, as often users do not want to share their personal schedules with other users or third-parties. Previous research has resulted in solutions that provide privacy guarantees, but they are either too complex or do not fit well in the popular user-provider operational model. In this paper, we propose practical and privacy-preserving solutions to the server-based scheduling problem. Our novel algorithms take advantage of the homomorphic properties of well-known cryptosystems in order to privately compute common user availabilities. We also formally outline the privacy requirements in such scheduling applications and we implement our solutions on real mobile devices. The experimental measurements and analytical results show that the proposed solutions not only satisfy the privacy properties but also fare better, in regard to computation and communication efficiency, compared to other well-known solutions.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Client/server; K.4.1 [**Public Policy Issues**]: Privacy

General Terms

Algorithms, Design, Performance, Security

Keywords

Activity scheduling, Client-server architecture, Homomorphic encryption

1. INTRODUCTION

Users rely increasingly on mobile devices such as smartphones and netbooks to access information while on the move [7], and very often they use the same equipment to store personal information about their daily schedules and activities [2]. Although many context and data sharing applications such as Google Maps, Facebook and Twitter are popular, activity management and synchronization applications are also gaining more and more attention [4]. Applications such as Microsoft Outlook [5], Apple iCal [1] and Nokia Ovi [6] are available on mobile devices and they all offer time and activity management services. One desirable feature in such applications is activity *scheduling*: colleagues can schedule meetings at common available time slots, groups of friends can organize parties on weekends and people unbeknownst to each other can engage in dating based on their common free/busy hours.

One concern in such scheduling applications is that users would prefer not to share all personal information with everyone. For example, they may only want to share common availabilities, but not details about other records. They may also have reservations about sharing personal information with third-party service providers. Therefore, privacy of personal information, *vis-à-vis* service providers and peers, is paramount for the success of such scheduling applications. For instance, a well-known service that allows users to find all common availabilities is Doodle [3]. However, Doodle does not provide privacy: Each user and the doodle server see the free/busy state of every user, and the private information that is leaked to all users and the central server is well beyond just the common available slots. Cultural, religious and many other private information can be easily inferred from availability patterns. Even if pseudonyms are used instead of real names, the server and all peers still know what time slots are available for everyone and how many users are free or busy.

Privacy-preserving scheduling problems have been extensively studied in the past by researchers from the theoretical perspective, for instance, by modeling them as set intersection problems [20, 11], distributed constraint satisfaction problems [27, 28, 24, 25], secure multi-party computation

problems [18, 12] and by framing them in the e-voting context [19]. Traditionally, there are two possible approaches to the scheduling problems: distributed and centralized. Distributed solutions do not rely on a third-party provider (and thus they prevent revealing information to the provider), but have several limitations. For instance, due to the frequent and intensive message exchanges among peers, scalability and computational complexity is an issue when dealing with a large number of (resource-limited) mobile devices; moreover, the need of sequencing among peers and the unpredictability of scheduling results (if a user interrupts the protocol) are two additional drawbacks. The centralized approaches, such as cloud-based computing, are better in terms of scalability, communication cost, complexity, synchronization and resilience but usually do not provide privacy, because users are required to transmit their personal information to the provider.

Our goal is to provide simple, practical and feasible solutions to the scheduling problem which, in addition to ensuring reasonable privacy guarantees, are easily integrated with existing operational models and mobile service providers. In this paper, we follow a centralized approach for addressing the problem of efficient and privacy-preserving scheduling. In the proposed schemes, users are able to determine common time slots without revealing any other information to either the other participants or to the central scheduling server. Our specific contributions are as follows. First, by building on the work of authors in related domains, we formally define the basic privacy requirements for users in a scheduling scenario. Second, we propose three novel privacy-preserving scheduling algorithms that take advantage of the homomorphic properties of asymmetric cryptosystems. Third, we implement the proposed algorithms on real mobile devices and perform extensive experiments using these devices in order to verify their computation and communication overheads. Finally, we explain how the system can be further made resilient to collusion and other well-known active attacks. To the best of our knowledge, we believe this is the first implementation and extensive testing of privacy-preserving scheduling schemes on commercial mobile devices.

The paper is organized as follows. We introduce the state-of-the-art in Section 2 and the system model and problem definition in Section 3. We formalize the privacy requirements for the scheduling problem in Section 4 and outline our algorithms in sections 5, 6 and 7. We present a comparative analysis and implementation results in Section 8, and we discuss the extensions of our schemes in Section 9. We conclude the paper in Section 10.

2. STATE OF THE ART

In the literature, the four most relevant bodies of work that address privacy in scheduling or similar scenarios are based on techniques from private set-intersection [20, 11], distributed constraint satisfaction [27, 28, 24, 25], secure multi-party computation [18, 12] and e-voting [19]. Hereafter, we review the most relevant aspects of such approaches.

In the private set-intersection domain, Kissner and Song [20] use mathematic properties of polynomials to design privacy-preserving union, intersection and element reduction operations on private multisets by leveraging on the Goldwasser-Micali homomorphic encryption scheme [17]. De Cristofaro and Tsudik [11] provide efficient variations of private-set intersection protocols and present a comparison in terms of computational and communication complexity, adversarial model and privacy. The authors also give informal definitions of client and server privacy. However, PSI approaches are generally distributed, and an efficient extension to an n-party protocol is challenging. In the meeting scheduling scenario, for instance, a trivial extension of the 2-party PSI to n parties (by running a 2-party protocol between each pair of users) would undermine the privacy of users' schedules as well; knowing the personal availability and the aggregate availability is sufficient to infer the other party's schedule.

Distributed constraint satisfaction approaches were investigated by Wallace and Freuder [27]: they study the tradeoff between privacy and efficiency and show that the information that entities learn during the negotiation of a common schedule has, in some cases, a tremendous impact on privacy. Details of an accept/reject response are exploited by intelligent agents in order to successfully infer the availabilities of other peers involved in the scheduling process. Similarly, Zunino and Campo [29] design a scheduling system in which entities learn and refine their knowledge about user preferences by using a Bayesian network. Yokoo *et al.* [28] use secret sharing among third-party servers in order to determine a suitable agreement among entities in a collusion-resistant way.

Solutions based on secure multi-party computation were investigated in [12] and a practical scheme was proposed in [18]. Herlea *et al.* [18], for instance, design and evaluate a distributed secure scheduling protocol by relying on properties of the XOR operation over binary values, in which all users contribute to the secrecy of individual schedules while ensuring the correctness of the results. Although not a pure e-voting scheme, Kellerman and Böhme [19] proposed an event scheduling protocol that inherits several security and privacy requirements from the e-voting context. However, a formal study of such properties and experimental performance results are missing in their work.

In contrast to most of the above solutions, we take a more centralized approach (with a single third-party server) for the privacy-preserving scheduling problem. Our solutions overcome communication and computational complexities intrinsic to most distributed approaches discussed above, as well as ensure that no private information (other than the resulting common availabilities) is exposed. Moreover, our protocols can easily fit into today's popular provider-consumer service architectures without incurring a huge communication cost on the service-provider.

3. SYSTEM MODEL

In this section, we outline the network and adversary model and formally define the scheduling problem.

3.1 Network Model

We assume that there is a total of N users u_i, $i \in \{1 \dots N\}$, that want to schedule an activity (meeting, party) at a common available time slot. Each user has a private schedule x_i represented by a string of bits $x_i = [b_{i,1}, b_{i,2}, \dots, b_{i,m}]$, where each bit $b_{i,j} \in \{0, 1\}$ expresses the availability of user

u_i in a particular time slot j; $b_{i,j} = 1$ means that user u_i is available at time slot j, whereas $b_{i,j} = 0$ means that the user is not available.[1] We assume that the length m of x_i, i.e. the time horizon of the individual schedules, is constant for all users. The value of m can either be predecided by the participants or fixed by the application.

Moreover, we assume that each user's device is able to perform public key cryptographic operations and that there is a semi-honest [16] (as detailed in Section 3.2) third-party performing the scheduling computations. The latter must be able to communicate with the users and run public key cryptographic functions as well. For instance, a common public-key infrastructure using the RSA [23] cryptosystem could be employed. All communications between a user and the third-party server will be encrypted with the latter's public key for the purposes of confidentiality of the schedules with respect to other users, for authentication and integrity protection. Thus, all users know the public key of the server but nobody, except the server, knows the corresponding private key. For simplicity of exposition, in our algorithms we do not explicitly show the cryptographic operations involving the server's public/private key.

We assume that the N users share a common secret, which is used to derive (i) a fresh common key pair (K_P, K_s), where K_p is the public key and K_s is the private key, and (ii) a fresh bit permutation function $\sigma = [\sigma_1, \ldots, \sigma_m]$ before initiating the scheduling operation. This could be achieved, for example, through a secure credential establishment protocol [9, 10, 21]. Thus, these keys and permutations are derived and known to each member of the group but not to the server. We refer to the encryption of a message M with the group public key as $E_{K_P,r}(M) = C$, where r is a random integer that is eventually needed, and to the decryption of the encrypted message C as $D_{K_s}(C) = M$. The permutation σ, although not strictly required, is used in order to randomize the order of bits sent to the server. This prevents the server from gaining any knowledge about which time slot is being evaluated in each computation.

3.2 Adversarial Model

Server

The third-party server is assumed to execute the scheduling protocols correctly, but it tries to learn any information it can from the input it gets by the users and the computations it performs. The server can accumulate the knowledge about users in each computation it performs. We refer to this adversarial behavior as *semi-honest*. More details about the semi-honest model can be found in [16].

Users

Users also want to learn private information about other users' schedules and, in addition to the passive eavesdropping attacks, users could act maliciously by generating fake users, manipulating their own schedules or by colluding with other users or the scheduling server. Initially, we assume that users are honest but curious (or semi-honest), and afterwards we present more active (or malicious) types of user adversaries in Section 9.2.

Although the semi-honest adversarial model is sufficient in most practical settings, considering the commercial interest of service providers and the mutual trust among participants, it does not include possible malicious behavior by the server or users. For instance, the server could collude with the participants or generate fake participants in order to obtain private information of the participants. Similarly, users might collude with other users or try to maliciously modify their schedules in order to disrupt the execution of the protocol or to gain information about other users' schedules. We address such active attacks by both users and server in Section 9.2, and we describe how such attacks can be thwarted by using existing cryptographic mechanisms.

3.3 Centralized Scheduling Algorithm

Given a group of N users $u_i, i \in \{1 \ldots N\}$, each with private schedules $x_i = [b_{i,1}, \ldots, b_{i,m}]$, the scheduling problem is to find time slots j such that $\forall i = 1 \ldots N$, $b_{i,j} = 1$, i.e. all users are available in the same time slot j. We refer to an algorithm that solves the scheduling problem as a *scheduling algorithm*. Formally, a scheduling algorithm A accepts the following inputs and produces the respective outputs:

- Input: a transformation of individual schedules
$$f(b_{i,1}, \ldots, b_{i,m}), \quad \forall i = 1 \ldots N.$$
where f is a transformation function such that it is hard (success with only a negligible probability) to determine the input of the function by just observing the output.
- Output: a function $g(Y), Y = y^1, \ldots, y^j, \ldots, y^m$ where:
$$y^j = \begin{cases} YES & \text{if } b_{i,j} = 1, \quad \forall i = 1 \ldots N \\ NO & \text{otherwise} \end{cases}$$

such that each user is able to compute $Y = g^{-1}(g(Y))$ using its local data. As we will see later on, we use the well-known cryptosytems ElGamal [13], Paillier [22] and Goldwasser-Micali [17] as our transformation and output functions f and g.

A centralized scheduling process works as follows. Each user $u_i, i \in \{1 \ldots N\}$ computes $f_i = f(b_{i,1}, \ldots, b_{i,m})$ and sends it to the third-party server, which then executes the scheduling algorithm A on the received inputs f_i, $\forall i$, and produces $g(Y) = A(f_1, \ldots, f_N)$. Finally, the server sends $g(Y)$ to each user who then obtains $Y = g^{-1}(g(Y))$. Figure 1 shows one execution of a generic centralized scheduling process.

4. PRIVACY DEFINITIONS

As mentioned earlier, in this paper we follow a centralized approach to solve the privacy-preserving scheduling problem. In other words, we assume that a third-party, given users' individual private schedules, computes their common

[1] In general, however, users may assign not only a binary value (available or busy) for each time slot, but they could express preferences [14, 15]. For example, $b_{i,j} \in 0, \ldots, 10$ where $b_{i,j} = 0$ means that user u_i is busy in the time slot j, whereas its preference would increase if $b_{i,j} \geq 1$. For simplicity of exposition, we assume a binary value here. We later discuss a more general case with non-binary costs in Section 9.

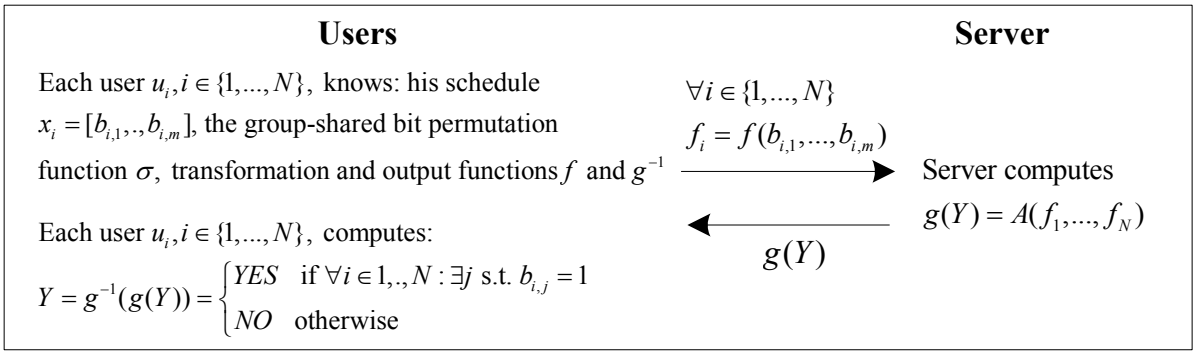


Figure 1: A generic scheduling protocol. Users first send their transformed schedules f_i to the server, which then performs the scheduling algorithm A on the received data and sends the encrypted output $g(Y)$ back to each user.

availabilities (time slots). /The privacy provided by a centralized scheduling algorithm can be defined in terms of the following two components: a) User-privacy and b) Server-privacy. Hereafter, we formally define each of these components. The symbols used throughout the paper are summarized in Table 1.

User-privacy

The *user-privacy* of any centralized scheduling algorithm A measures the probabilistic advantage that any user $u_i, i \in \{1 \dots N\}$ gains towards learning the private schedules of at least one other user $u_j, j \neq i$, except their common availabilities, after all users have participated in the execution of the algorithm A. In order to accurately measure users' privacy, we need to compute the following two advantages. First, we measure the *Identifiability Advantage*, which is the probabilistic advantage of an adversary in correctly guessing a schedule bit (which is not a common availability) of any other user. We denote it as $Adv_{u_i}^{IDT}(A)$. Second, we measure the *Linkability Advantage*, which is the probabilistic advantage of an adversary in correctly guessing that any two or more other users have exactly the same corresponding schedule bit (not a common availability bit) without necessarily knowing the values of those bits. We denote this advantage as $Adv_{u_i}^{LNK}(A)$. We make the following straightforward observation.

Observation 1. If an adversary has identifiability advantage over two corresponding schedule bits of two different users, this implies that it has linkability advantage over those two bits as well. However, the inverse is not necessarily true.

We semantically define the identifiability and linkability advantages using a challenge-response methodology. Challenge-response games have been widely used in cryptography to prove the security of cryptographic protocols. We now describe such a challenge-response game for the identifiability advantage $Adv_{u_i}^{IDT}(A)$ of any user u_i participating in the algorithm A as follows.

1. Initialization: Challenger privately collects $x_i = [b_{i,1}, \dots, b_{i,m}]$ and $f_i = f(b_{i,1}, \dots, b_{i,m})$ from all users $u_i, i \in \{1 \dots N\}$.

Table 1: Table of symbols.

SYMBOL	DEFINITION
$Adv^{LNK}(A)$	Linkability advantage
$Adv^{IDT}(A)$	Identifiability advantage
$D(C)$	Decryption of a ciphertext C
$E_{K,r}(m)$	Encryption of a message m using the key K and a random number r
K_P	Shared public key of the N users
K_S	Shared private key of the N users
m	Number of slots of each individual schedule
N	Number of users
$x_i=[b_{i,1},..,b_{i,m}]$	Schedule of user u_i, where $b_{i,j}$ is the availability at time slot j
$\sigma=[\sigma_1,..,\sigma_m]$	Schedule permutation function

2. Scheduling: Challenger computes $g(Y) = A(f_1, f_2, \dots, f_N)$ with the users and sends $g(Y)$ to all users $u_1, u_2, \dots, u_N$.
3. Challenger randomly picks a user u_i, $i \in \{1 \dots N\}$, as the adversary.
4. u_i picks $j \in \{1 \dots N\}$, *s.t.* $j \neq i$ and sends it to the challenger.
5. Challenge: the challenger picks a random time slot $p \in \{1 \dots m\}$, *s.t.*, $\exists b_{k,p} = 0$ for at least one $k \in 1, \dots, N$. Challenger then sends (j, p) to the user u_i. This is the challenge.
6. Guess: User u_i sends $b'_{j,p} \in \{0, 1\}$ to the challenger as a response to his challenge. If $b'_{j,p} = b_{j,p}$, the user u_i (adversary) wins; otherwise, he loses.

The identifiability advantage $Adv_{u_i}^{IDT}(A)$ can be defined as

$$Adv_{u_i}^{IDT}(A) = \left| Pr_{u_i}[b'_{j,p} = b_{j,p}] - \frac{1}{2} \right| \quad (1)$$

where $Pr_{u_i}[b'_{j,p} = b_{j,p}]$ is the probability of user u_i winning the game (correctly answering the challenge in the challenge-response game), computed over the coin flips of the challenger, $b'_{j,p}$ is u_i's guess about the schedule of user u_j in

the time slot p and $b_{j,p}$ is u_j's true availability. An external attacker, having no access to the output of the algorithm, has obviously no advantage at all. Thus, we focus on the non-trivial case with participating users only.

Similarly, we describe the challenge-response game for the linkability advantage $Adv_{u_i}^{LNK}(A)$ of any user u_i as follows.

1. Initialization: Challenger privately collects $x_i = [b_{i,1}, \ldots, b_{i,m}]$ and $f_i = f(b_{i,1}, \ldots, b_{i,m})$ from all users $u_i, i \in \{1 \ldots N\}$.
2. Scheduling: Challenger computes $g(Y) = A(f_1, f_2, \ldots, f_N)$ with the users and sends $g(Y)$ to all users $u_1, u_2, \ldots, u_N$.
3. Challenger randomly picks a user u_i, $i \in \{1 \ldots N\}$, as the adversary.
4. u_i picks $h, j \in \{1 \ldots N\}$, *s.t.* $j \neq h, j \neq i, h \neq i$ and sends (h, j) to the challenger.
5. Challenge: Challenger randomly picks a time slot $p \in \{1 \ldots m\}$, *s.t.*, $\exists b_{k,p} = 0$ for at least one $k \in 1, \ldots, N$. Challenger then sends (j, p) and (h, p) to the user u_i. This is the challenge.
6. Guess: User u_i decides if $b_{j,p} = b_{h,p}$ or not. User u_i sets $b' = 1$ if he decides $b_{j,p} = b_{h,p}$ and $b' = 0$ if he decides $b_{j,p} \neq b_{h,p}$. User u_i sends b' to the challenger as a response to his challenge. If $b_{j,p} = b_{h,p}$ and $b' = 1$ or if $b_{j,p} \neq b_{h,p}$ and $b' = 0$, the user u_i (adversary) wins; otherwise, he loses.

The linkability advantage $Adv_{u_i}^{LNK}(A)$ can be defined as

$$Adv_{u_i}^{LNK}(A) = |Pr_{u_i}[((b_{j,p} = b_{h,p}) \ \wedge \ b' = 1) \vee ((b_{j,p} \neq b_{h,p}) \ \wedge \ b' = 0)] - \frac{1}{2}| \quad (2)$$

where $Pr_{u_i}[.]$ is the probability of user u_i winning the game, computed over the coin flips of the challenger. As for the identifiability advantage, an external attacker has no linkability advantage at all.

We now define the user-privacy of the scheduling algorithm A on a per-execution basis as follows:

Definition 1. An execution of the centralized scheduling algorithm A is *user-private* if both the identifiability advantage $Adv_{u_i}^{IDT}(A)$ and the linkability advantage $Adv_{u_i}^{LNK}(A)$ of each participating user $u_i, i \in \{1, \ldots, N\}$ is negligible.

A function $f(x)$ is called *negligible* if, for any positive polynomial $p(x)$, there is an integer B such that for any integer $x > B$, $f(x) < 1/p(x)$ [16].

Definition 1 says that a particular execution of the scheduling algorithm is user-private if and only if users do not gain any (actually, negligible) additional knowledge about the schedule bits of any other user, except the schedule bits that have a value 1 for all users (common availabilities).

Server-privacy

The *server-privacy* of any (centralized) scheduling algorithm A measures the probabilistic advantage that the server (which executes the scheduling algorithm A and observes the inputs from the users) gains towards learning the private schedules of at least one user $u_i, i \in \{1 \ldots N\}$. As in the case of user-privacy, we need to compute the following two advantages. First, the advantage of the server in guessing correctly any schedule bit of any user participating in the scheduling algorithm, called as *Identifiability Advantage* and denoted as $Adv_S^{IDT}(A)$. Second, the advantage of the server in guessing correctly that any two (or more) participating users have exactly the same corresponding schedule bits without necessarily knowing the values of those bits, called the *Linkability Advantage* and denoted as $Adv_S^{LNK}(A)$.

The server identifiability and linkability advantages are defined in a similar fashion as the user advantages. The challenge-response game for the server identifiability advantage $Adv_S^{IDT}(A)$ is defined as follows.

1. Initialization: Challenger privately collects $x_i = [b_{i,1}, \ldots, b_{i,m}]$ and the server privately collects $f_i = f(b_{i,1}, \ldots, b_{i,m})$ from all users $u_i, i \in \{1 \ldots N\}$.
2. Scheduling: Server computes $g(Y) = A(f_1, f_2, \ldots, f_N)$ with the users and sends $g(Y)$ to all users $u_1, u_2, \ldots, u_N$.
3. Server picks $i \in \{1 \ldots N\}$ and sends it to the challenger.
4. Challenge: Challenger randomly picks a time slot $p \in \{1 \ldots m\}$. Challenger then sends (i, p) to the server. This is the challenge.
5. Guess: server sends $b'_{i,p} \in \{0, 1\}$ to the challenger as a response to his challenge. If $b'_{i,p} = b_{i,p}$, the server (adversary) wins; otherwise, he loses.

The identifiability advantage $Adv_S^{IDT}(A)$ is defined as

$$Adv_S^{IDT}(A) = \left| Pr_S[b'_{j,p} = b_{j,p}] - \frac{1}{2} \right| \quad (3)$$

where $Pr_S[b'_{j,p} = b_{j,p}]$ is the probability of the server winning the game, computed over the coin flips of the challenger.

The challenge-response game for the server linkability advantage $Adv_S^{LNK}(A)$ is defined as follows.

1. Initialization: Challenger privately collects $x_i = [b_{i,1}, \ldots, b_{i,m}]$ and the server privately collects $f_i = f(b_{i,1}, \ldots, b_{i,m})$ from all users $u_i, i \in \{1 \ldots N\}$.
2. Scheduling: Server computes $g(Y) = A(f_1, f_2, \ldots, f_N)$ with the users and sends $g(Y)$ to all users $u_1, u_2, \ldots, u_N$.
3. Server picks $h, j \in \{1 \ldots N\}$, *s.t.* $j \neq h$ and sends (h, j) to the challenger.
4. Challenge: Challenger randomly picks $p \in \{1 \ldots m\}$ and then sends (j, p) and (h, p) to the server. This is the challenge.

5. Guess: Server decides if $b_{j,p} = b_{h,p}$ or not. Server sets $b' = 1$ if he decides $b_{j,p} = b_{h,p}$ and $b' = 0$ if he decides $b_{j,p} \neq b_{h,p}$. Server sends b' to the challenger as a response to his challenge. If $b_{j,p} = b_{h,p}$ and $b' = 1$ or if $b_{j,p} \neq b_{h,p}$ and $b' = 0$, the server (adversary) wins; otherwise, he loses.

The linkability advantage $Adv_S^{LNK}(A)$ is defined as

$$Adv_S^{LNK}(A) = |Pr_S[(b_{j,p} = b_{h,p}) \ \wedge \ b' = 1) \vee \ (b_{j,p} \neq b_{h,p}) \ \wedge \ b' = 0)] - \frac{1}{2}| \quad (4)$$

where $Pr_S[.]$ is the probability of the server winning the game, computed over the coin flips of the challenger.

The server-privacy of the scheduling algorithm A on a per-execution basis can then be defined as follows:

Definition 2. An execution of the centralized scheduling algorithm A is *server-private* if both the identifiability advantage $Adv_S^{IDT}(A)$ and the linkability advantage $Adv_S^{LNK}(A)$ of the server is negligible.

Now, it is reasonable to assume that in practice users will be able to perform multiple executions of the scheduling algorithm with possibly different participating sets of users. This is especially true if such an algorithm is offered, for example, as a service by mobile service providers to their subscribers. Thus, privacy of the scheduling algorithm should be defined over multiple executions. First, we define a *private execution* as follows:

Definition 3. A *private* execution is an execution which does not reveal more information than what can be derived from its result and the prior knowledge.

Based on how memory is retained over sequential executions, we define two types of algorithm executions, namely, independent and dependent:

Definition 4. An *independent (respectively, dependent) execution* is a single private execution of the scheduling algorithm defined in Section 3.3 in which *no (respectively, some)* information of an earlier and current execution is retained and passed to future execution.

The information retained can include past inputs to the algorithm, intermediate results (on the server) and the outputs of the algorithm. Based on the type of executions, we define a privacy-preserving scheduling algorithm as follows:

Definition 5. A scheduling algorithm A is *execution* (respectively *fully*) *privacy-preserving* if and only if for every *independent* (respectively *all*) execution(s):

1. A is correct; All users are correctly able to compute $y^j = 1, \forall j = 1 \ldots m$ if and only if $b_{i,j} = 1, \forall i = 1 \ldots N$.
2. A is user-private in every execution.
3. A is server-private in every execution.

A fully privacy-preserving algorithm is a much stronger (and difficult to achieve) privacy requirement. In this work, similar to earlier efforts, we focus on achieving execution privacy. The following observation gives the relationship between fully privacy-preserving and execution privacy-preserving scheduling algorithms.

Observation 2. Any scheduling algorithm A, as defined in Section 3.3, is execution privacy-preserving if it is fully privacy-preserving, but the inverse is not true.

Next, we outline our centralized scheduling algorithms.

5. SCHEDELG ALGORITHM

In this section, we describe our first privacy-preserving centralized scheduling scheme, which is based on the ElGamal [13] cryptosystem. The security of the ElGamal encryption relies on the intractability of the discrete logarithm problem (DLP), which assumes that it is computationally infeasible to obtain the private key K_s given the public key (g, h), where g is a generator of a multiplicative cyclic group G of prime order q and $h = g^{K_s} \mod q$.

Our protocol *SchedElG* uses the *homomorphic* property of the ElGamal cryptosystem in order to allow the scheduling server to compute the aggregated availabilities by working only on the encrypted individual schedules. For instance, it can be verified that the ElGamal scheme satisfies:

$$D(E_{K_P,r_1}(m_1) \cdot E_{K_P,r_2}(m_2)) =$$
$$D((g^{r_1}, m_1 h^{r_1}) \cdot (g^{r_2}, m_2 h^{r_2})) =$$
$$D(g^r, (m_1 \cdot m_2) h^r) = m_1 \cdot m_2$$

where $r = r_1 + r_2 \in \mathbb{Z}_q$ is a random integer. Moreover, being a probabilistic encryption scheme, it follows that if $r_1 \neq r_2$, $E_{K_P,r_1}(m) \neq E_{K_P,r_2}(m)$.

For the *SchedElG* algorithm, we assume that the meeting participants represent their availabilities in the following way: $b^*_{i,j} = 1$ if $b_{i,j} = 1$, but $b^*_{i,j} = R$ (where $R \in \mathbb{Z}_q, R > 1$ is a random integer) if $b_{i,j} = 0$.

Scheme

The privacy-preserving scheduling protocol *SchedElG* is shown in Figure 2. All users first select the sequence of time slots according to the permutation σ, i.e., $\sigma_j, \forall j = 1..m$, and then encrypt individually the corresponding schedule availabilities, i.e., $E_i = [E_{i,\sigma_1}, \ldots, E_{i,\sigma_m}]$ where $E_{i,\sigma_j} = E_{K_P,r_{i,j}}(b^*_{i,\sigma_j})$. Then, each user sends its E_i privately to the scheduling server that performs the multiplication $\prod_{i=1}^{N} E_{i,\sigma_j}$ of all users' encrypted schedules E_{i,σ_j}, for $j = 1, \ldots, m$. The results of such operation are the (encrypted) aggregated availabilities of all users for each time slot j. Next, the server replies with the aggregated encrypted result E_{sched} back to each user. Each slot in E_{sched} contains a product of the individual time-slot bits encrypted with the users' shared key. Finally, each user decrypts the result and obtains the aggregated availabilities $[y^1 = B^*_{\sigma_1}, \ldots, y^m = B^*_{\sigma_m}]$ of all users u_i

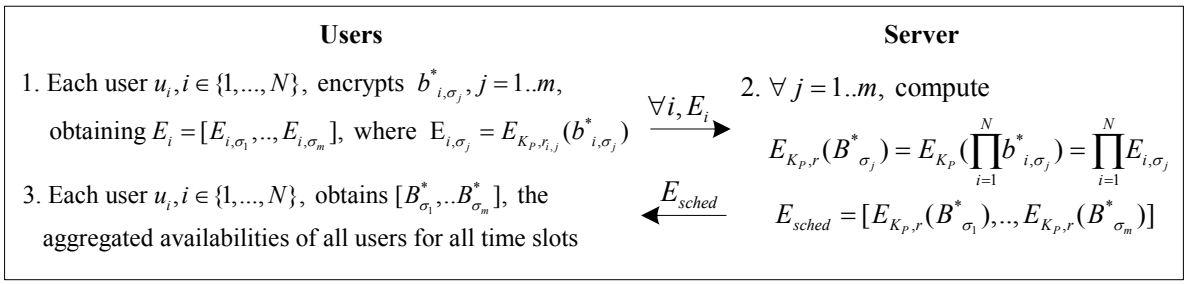


Figure 2: *SchedElg* protocol.

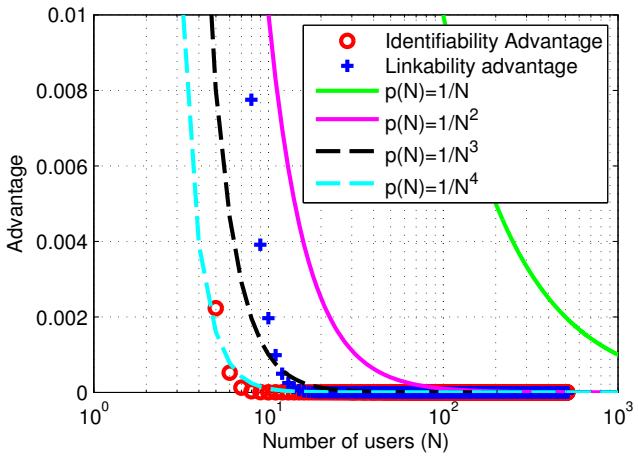


Figure 3: Identifiability and linkability advantages of an adversary.

for each time slot σ_j. If $B^*_{\sigma_j} = 1$, it means that all users are available at time slot σ_j; if $B^*_{\sigma_j} > 1$, then at least one user is not available and therefore σ_j is not a suitable time slot. The following result shows the correctness and privacy properties of *SchedElG*.

LEMMA 1. *The protocol* SchedElG *is correct and execution privacy-preserving.*

Due to space constraints, we omit the proof of Lemma 1 here. Interested readers can find the proof in the full paper [8]. In Figure 3, we plotted the identifiability and linkability advantages of an adversary for *SchedElg*, compared with polynomially (in terms of the number of participants N) decreasing functions $p(N)$, for increasing values of N. As confirmed by our analysis, the plot shows that there is always an integer N such that for any integer $x > N$, the identifiability and linkability advantages are smaller than $1/p(x)$.

6. SCHEDPA ALGORITHM

In this section, we define our second privacy-preserving scheduling scheme, which is based on the Paillier cryptosystem [22]. The security of the Paillier encryption scheme is based on the intractability of determining whether an integer r is an n-residue mod n^2, where n is a composite number. In our protocol, we use the homomorphic properties of the Paillier cryptosystem to compute in a privacy-preserving fashion the availability of all users involved in the scheduling process. In particular, one can verify that the Paillier scheme satisfies the following:

$$D[E_{K_P,r_1}(m_1) \cdot E_{K_P,r_1}(m_2) \mod n^2] = m_1 + m_2 \mod n$$
$$D[E_{K_P,r}(m_1)^{m_2} \mod n^2] = m_1 \cdot m_2 \mod n$$

where $r_i, r \in \mathbb{Z}_n^*$ are random numbers chosen by the encrypters, $m \in \mathbb{Z}_n$ is the message to encrypt and $n = pq$ where p, q are two large primes. The randomness in the encryption ensures that if $r_1 \neq r_2$, $E_{K_P,r_1}(m) \neq E_{K_P,r_2}(m)$.

To adapt our scheme to the addition property of Paillier's homomorphism, we take the bit value $\bar{b}_{i,j}$ in the computation instead of the original bit value $b_{i,j}$ as follows: $\bar{b}_{i,j} = 0$ if $b_{i,j} = 1$, and $\bar{b}_{i,j} = r$ (where $r \in \mathbb{Z}_n^*, r > 1$ is a random integer) if $b_{i,j} = 0$.

Scheme

The corresponding privacy-preserving scheduling protocol is shown in Figure 4. First, all users select the sequence of time slots according to the permutation σ, i.e., σ_j, $\forall j = 1,\ldots,m$, and then encrypt individually the corresponding availabilities, i.e. $E_i = [E_{i,\sigma_1},\ldots,E_{i,\sigma_m}]$ where $E_{i,\sigma_j} = E_{K_P,r_{i,j}}(\bar{b}_{i,\sigma_j})$. Then, each user sends its E_i privately to the scheduling server that performs the multiplication and exponentiation $(\prod_{i=1}^{N} E_{i,\sigma_j})^R$ of all users' encrypted schedules E_{i,σ_j}, for $j = 1,\ldots,m$, in order to obtain the encryption of the value V_{σ_j} that is needed by the users. Afterwards, the server sends the aggregated encrypted result E_{sched} back to each user. Each slot in E_{sched} contains a randomly scaled sum of the individual time-slot bits $\bar{b}_{i,\sigma_j}$ encrypted with the users' shared key. Finally, each user decrypts the result and knows that if $V_{\sigma_j} = 0$, the time slot σ_j is available for everybody. If $V_{\sigma_j} > 1$, then at least one user is not available. Note that even if the server chooses $R = 1$, the privacy of the users is preserved with $\bar{b}_{i,j}$. The following result shows the correctness and privacy properties of *SchedPa*.

LEMMA 2. *The protocol* SchedPa *is correct and execution privacy-preserving.*

Due to space constraints, we omit the proof of Lemma 2 here. Interested readers can find the proof in the full paper [8].

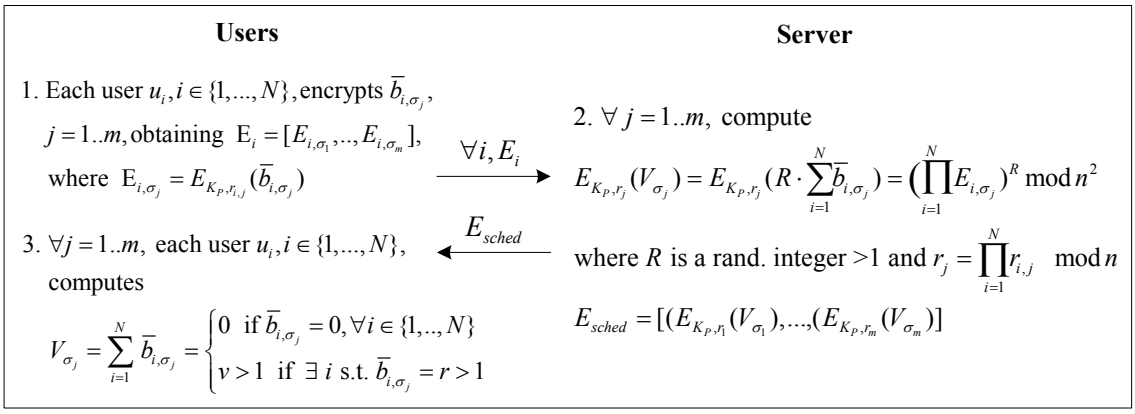


Figure 4: *SchedPa* protocol.

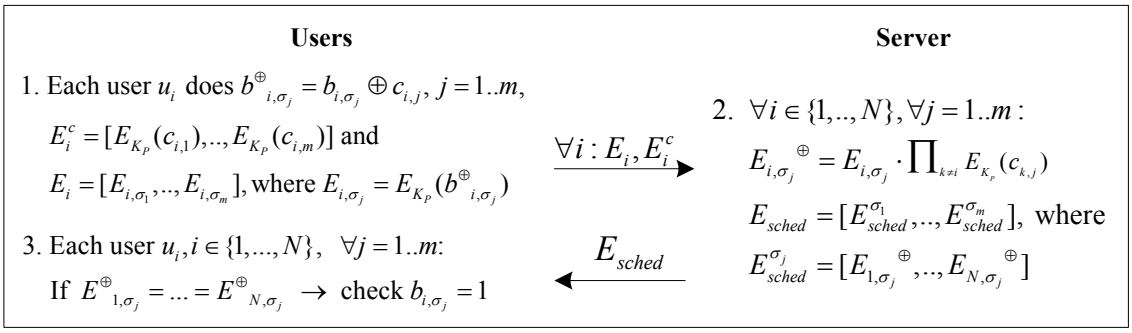


Figure 5: *SchedGM* protocol.

7. SCHEDGM ALGORITHM

In this section, we present our last privacy-preserving scheduling algorithm, which is based on the Goldwasser-Micali (GM) cryptographic scheme [17]. The security of the GM encryption relies on the intractability of the quadratic residuosity problem, i.e. on the infeasibility of determining whether or not an integer r is a quadratic residue mod n when the Jacobi symbol for r is 1, given $n = pq$ where p, q are large primes. *SchedGM* makes use of the following homomorphic property of the GM cryptosystem:

$$D[E_{K_P,r_1}(m_1) \cdot E_{K_P,r_2}(m_2)] = m_1 \oplus m_2$$

The intuition behind the protocol is based on the work by Herlea *et al.* [18], in which users privately establish a global bit mask (unknown to any user) and then compare all the masked availabilities without knowing the true bit value b_{i,σ_j} of the other users. If all users have the same masked bit value for a given time slot σ_j, then each user knows that everybody else has the same availability, which can be inferred by looking at the private unmasked bit value b_{i,σ_j}. Although initially used in a distributed scenario, we extend the general idea to the centralized scheme as well.

Assumption

Each user u_i generates a private random bit mask $s_i = [c_{i,1}, c_{i,2}, \ldots, c_{i,m}], c_{i,j} \in \{0,1\}$, of the same length of the schedule x_i.

Scheme

The privacy-preserving scheduling algorithm is shown in Figure 5. Each user first selects the sequence of time slots according to the permutation σ, i.e., σ_j, $\forall j = 1, \ldots, m$, and then masks the corresponding schedule bits, i.e. $b^{\oplus}_{i,\sigma_j} = b_{i,\sigma_j} \oplus c_{i,j}$. Then, each user encrypts individually both its bit mask, i.e. $E_i^c = [E_{K_P,r_{i,1}}(c_{i,1}), \ldots, E_{K_P,r_{i,m}}(c_{i,m})$, and the masked availabilities, i.e. $E_i = [E_{i,\sigma_1}, \ldots, E_{i,\sigma_m}]$, where $E_{i,\sigma_j} = E_{K_P,r_{i,j}}(b^{\oplus}_{i,\sigma_j})$. Afterwards, each user u_i sends its E_i and E_i^c to the server, which computes the multiplication of the received E_{i,σ_j} with the encrypted masks of all other users $u_k, \forall k \neq i$, obtaining $E^{\oplus}_{i,\sigma_j} = E_{i,\sigma_j} \cdot \prod_{k \neq i} E_{K_P}(c_{k,j})$, $\forall i \in 1, \ldots, N$ and $\forall j = 1, \ldots, m$. Afterwards, the server sends all individual schedules, masked by a global mask $c_{1,j} \oplus \ldots \oplus c_{N,j}$, to each user in a random order. As a result, a user will not know his own schedule (masked with the global mask), otherwise he would be able to determine the global mask. Finally, each user decrypts the received messages and compares all masked individual schedules. If for a given time slot σ_j they all have the same value, then each user u_i can infer whether the time slot σ_j is available by looking at its own schedule b_{i,σ_j}. The following result shows the correctness and privacy properties of *SchedGM*.

LEMMA 3. *The protocol* SchedGM *is correct and server-private.*

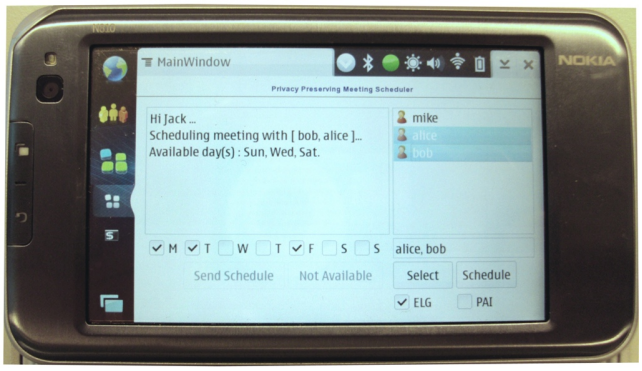


Figure 6: Frontend of the scheduling application on a Nokia N810.

Due to space constraints, we omit the proof of Lemma 3 here. Interested readers can find the proof in the full paper [8].

8. IMPLEMENTATION AND DISCUSSION

Before presenting the implementation details, let us first perform a comparative analysis of the asymptotic complexities of the proposed protocols, as shown in Table 2. In order to compare our three algorithms with an equivalent security, we set the bit-lengths of the ElGamal modulus q and the Paillier and GM modulus n to 1024 bits. A time-slot availability would then be encrypted to a 2-tuple of 1024-bit ciphertexts for ElGamal, to a 1024-bit ciphertext for GM and to a 2048-bit ciphertext for the Paillier encryption scheme.

From Table 2 we can see that the *SchedElG* and *SchedPa* protocols are very efficient, both in terms of communication $O(m)$, where m is the number of time slots, and computation complexity $O(m)$. Moreover, these two algorithms provide strong privacy guarantees. *SchedGM*, on the other hand, is comparatively less efficient due to the greater number of exchanged messages ($O(N \cdot m)$, where N is the number of participants). From the privacy perspective, *SchedGM* reveals more information: users can infer the ratio of free/busy participants for each time slot without identifying the ones that are busy and the ones that are free. Because in all schemes, the server operates only on encrypted data, it cannot gain any knowledge about the users' private schedules.

Distributed [24, 18] and hybrid [28] solutions proposed in the literature are less efficient from the communication standpoint as compared to the proposed protocols. Moreover, the computational complexity of these schemes is higher than *SchedElG* and *SchedPa*, and this undermines their applicability on resource-constrained mobile platforms. Even though the hybrid approach [28] has comparable computation complexity, it is not completely reliable from the privacy point of view because it assumes that the server(s) can get clear-text access to the individual availabilities.

We further evaluate the performance of *SchedElg*, *SchedPa* and *SchedGM* by implementing the client component of the protocols and primitives on Nokia N810 mobile devices with 400 MHz CPU and 128 MB RAM (Figure 6), and the server component on a desktop computer with 2 GHz CPU and 3 GB RAM. The results of the experimentation are shown in Figure 7 and 8.

Table 2: Efficiency and privacy of scheduling protocols (DisCSP [28], MPC-DisCSP2 [24] and SDC [18])

		Per-user encr.	*Per-user decr.*	*Per-user comm.*	*Order of an encr. availab.*	*Privacy properties*
Centralized	***SchedElG***	O(m)	O(m)	O(m)	1024 bits	User-private Server-private
	SchedPa	O(m)	O(m)	O(m)	2048 bits	User-private Server-private
	SchedGM	O(m)	O(N · m)	O(N · m)	1024 bits	User-private [#] Server-private
	Naïve	0	0	O(m)	1 bit *	None
Hybrid	**DisCSP protocol**	O(m)	O(m)	O(N · m)	1024 bits	Private
Distributed	**MPC-DisCSP2 protocol**	O(N · m)	O(m)	O(N · m)	2048 bits	Private
	SDC protocol	$O(N^2 \cdot m)$	O(N · m)	$O(N \cdot m \cdot \lceil \log_2(N) \rceil)$	1024 bits	Private

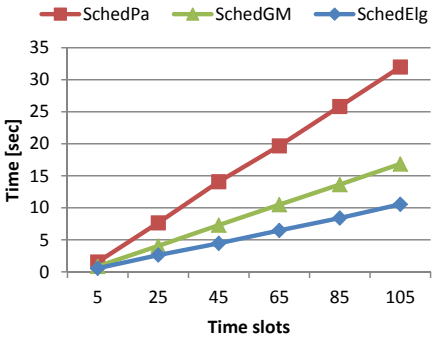
(*) The naïve algorithm does not encrypt the schedule bits
(#) Adv^{IDT} is a negligible function, whereas, for some output Y of the algorithm, Adv^{LNK} is non-negligible

Client encryption

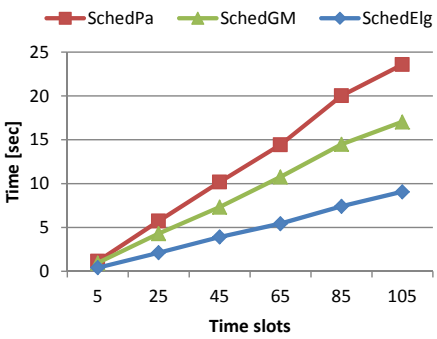
As we can see from Figure 7, the time required to perform the scheduling operations increases with the number of time slots for all the proposed algorithms, which is intuitive. With respect to encryption performance, Figure 7(a) shows that *SchedElg* is the most efficient scheduling algorithm, requiring 4 seconds to encrypt 45 time slots (a typical weekly schedule on a per hour basis). The same task is accomplished by *SchedGM* and *SchedPa* in 7 and 14 seconds respectively. These results might be explained by the following. First, the cryptographic primitives for the ElGamal scheme are implemented in a standard well-optimized library, *libgcrypt*, present in most Unix-based operating systems. *SchedGM*, on the contrary, does not use a standard library. We implemented the Goldwasser-Micali cryptosystem libraries, and as such it is likely that further optimization could significantly improve the performance. Second, the encrypted elements in *SchedPa* have twice the bit-length of the ones used in the other two algorithms, and therefore the same operations (multiplications and exponentiations) require more time.

Client decryption

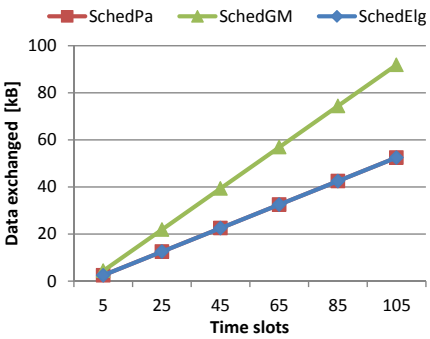
Figure 7(b) shows the time required for decrypting the final result (common availabilities) of the scheduling algorithms at the client. Similarly to the encryption time, the fastest algorithm for the decryption is *SchedElg*, which takes 4 seconds in order to obtain the aggregated availabilities for a 45 time-slot period. For the same number of time slots, *SchedPa* takes approx. 7 seconds, which is almost twice the best performance. The decryption times for both *SchedElg* and *SchedPa* are independent of the number of participants. The performance of *SchedGM*, due to the fact that the final output of the algorithm is a sequence of vectors instead of just a single aggregated vector, is decreasing with the number of users as well as with the number of time slots.



(a) Encryption performance



(b) Decryption performance. We considered $N = 5$ for *SchedGM*.



(c) Communication efficiency (at application layer). We considered $N = 5$ for *SchedGM*.

Figure 7: Client implementation performance.

Thus, for a reasonable number of participants (e.g. $N = 5$), *SchedGM* is still practical enough to be implemented on resource-constrained mobile devices, although it is not the preferred solution.

Client communication

Figure 7(c) shows the (application layer) data that each client exchanges during one execution of the scheduling algorithm. In general, all the proposed privacy-preserving scheduling algorithms have reasonable communication costs. *SchedElg* and *SchedPa* are the most efficient algorithms and they require 22 kB of data in order to compute the aggregated availabilities of a 45 time-slot period, whereas *SchedGM* requires 39 kB for the same result. As previously mentioned, *SchedGM* uses a sequence of masked vectors in order to compute the final availabilities of the users, and therefore the amount of data is proportional both to the number of users and time-slots.

Server performance

The scheduling server's performance is shown in Figure 8. As it can be seen, the time required to perform the scheduling operations on encrypted values increases with both the number of users and time slots, which is intuitive. Even with a large number of users and time slots, the amount of time required for the operations is still below 0.2 second, which suggests that the load on the server is limited, which allows it to efficiently handle multiple scheduling events, without incurring in huge computational overhead.

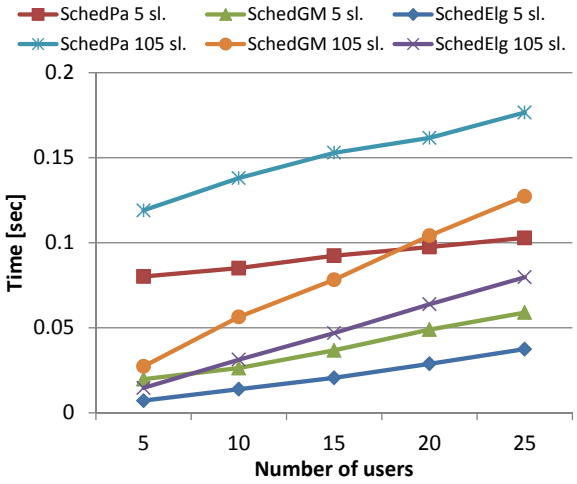


Figure 8: Server processing performance.

9. EXTENSIONS

In this section, we show how *SchedPa* can be easily extended to the case where user schedules are non-binary, i.e., each time slot is a non-negative cost $C_{i,j}$ that indicates u_i's preference for time-slot j. We also describe several active attacks on the proposed scheduling schemes, such as collusion between users-server and data modification by the users, and how these attacks can be mitigated by using existing cryptographic mechanisms. Finally, we discuss some further enhancements for the privacy of users' schedules and how to implement them.

9.1 Non-binary Schedules

The goal here is to find, in a privacy-preserving fashion, the time-slot with the minimum aggregated cost. The scheme works as follows:

1. Each user u_i reorders its cost sequence $C_{i,1} \dots C_{i,m}$ using the shared permutation σ and encrypts each cost C_{i,σ_j} in the sequence using the Paillier cryptosystem with the shared group key K_P. It then passes the result $(E_{K_P,r_{i,1}}(C_{i,\sigma_1}) \dots E_{K_P,r_{i,m}}(C_{i,\sigma_m}))$ to the server.

2. The server computes the encrypted sum of costs $E_{K_P,r_j}(R \cdot \sum_{i=1}^{N} C_{i,\sigma_j}), \forall j$, where R is a random integer greater than one chosen by the server.

3. The server selects a pre-determined user u_k and passes a *randomly ordered* (different from σ) sequence of the encrypted aggregated costs to it. This is to prevent u_k from learning the aggregated cost function.

4. User u_k decrypts all the elements passed from the server, and identifies the minimum aggregated cost.

5. User u_k then queries the server for the index of the (en-

crypted) minimum aggregated cost. The server then distributes the queried index to all users.

It can be easily shown that the above scheme is execution privacy-preserving. For conciseness, we do not discuss the details of the privacy analysis here.

9.2 Active Attacks

There are five kinds of possible active attacks on the scheduling schemes: (i) collusion between the scheduling server and users, (ii) collusion among users, (iii) fake user generation by the server, (iv) individual user schedule modification and (v) integrity and replay attacks.

In order to thwart the first issue, the invited participants could agree on establishing a shared secret using techniques from threshold cryptography, such as [26]. The server should then collude with at least a predefined number of participants in order to obtain the shared secret and learn the individual availabilities. The second concern may arise if k colluding users set their schedules to *all-available*, and try to learn the schedules of other users. Assuming that N is the total number of participants and k the number of colluding ones, our schemes would provide some level of schedule privacy to honest users as long as $N - k \geq 2$. Only if all but one users collude, then they are able to determine the schedule of the remaining user. In order for the third attack to succeed, the server would need to generate fake users and convince the true participants about the legitimacy of the fake users. In practice, this is a non-trivial task to achieve, and thus the attack has a very slim chance of succeeding. Moreover, the effectiveness of such attack could be further reduced by adopting the threshold cryptographic scheme mentioned previously, because the server would then need to generate k fake users and validate them as true participants.

The fourth attack is also not able to succeed in revealing the availability of other meeting participants, as the best a malicious user can do is to set its own schedule to all-available, and then guess the availabilities of the other $N-1$ participants. Even if a malicious user attempts to modify its own schedule with invalid values, such as negative values, the message domain restrictions of cryptosystems (such as ElGamal and Paillier) would prevent such modifications. Thus, malicious attacks consisting of manipulating the final result by using invalid negative values as schedule values are not possible in the proposed protocols.

The last attack concerns the integrity and freshness of the encrypted schedules. The participants are the only entities in the system that know the secret that has been used to generate the public/private key pair, and therefore they are the only ones that can generate and verify the integrity of the encrypted data. Moreover, using the shared common secret, each participant could generate a fresh *nonce* at each algorithm execution and send it (in encrypt form) to the server during the scheduling process. The server would then forward these encrypted nonces to each participant, who could verify that all received nonces are equal. If not all nonces are equal, then the participants know that there has been at least one replay attack, and thus the schedule results are not to be trusted.

9.3 Single Available Time Slot

The output of conventional, not privacy-preserving scheduling services (such as Doodle [3] or Outlook [5]) consists of time slots in which all participating users are available. The proposed schemes follow this paradigm and they provide, in an efficient and privacy-preserving way, all time-slots for which all users are available.

In some cases, however, it might be desirable to limit the disclosure of common availabilities to only one time-slot, instead of the set of all available time-slots. This would provide an additional layer of privacy for the individual schedules, as the participants would be given a single feasible solution. Hereafter we describe one simple way to adapt the proposed schemes to support this feature (Figure 9).

First, all users participating in the scheduling process perform step 1 of the respective algorithm (*SchedElg*, *SchedPa* or *SchedGM*). Second, the server performs step 2 but it does not send the final output to each user. Instead, it randomly chooses a private time-slot permutation function $\theta = [\theta_1, \ldots, \theta_m]$ and applies it to the elements of the final output vector(s) E_{sched}. We call this new vector(s) E^{θ}_{sched}. At this point, the schedules have been permuted twice, once by the users prior to the encryptions (with σ) and once by the server (with θ).

Next, the server sends E^{θ}_{sched} to the user who started the activity scheduling (the *initiator*), which then gets the common availabilities but in a doubly permuted order. The initiator is able to determine the *available* slots in this doubly permuted time slot list, but he is not able to determine the time slots they correspond to in the original schedule. The initiator selects one commonly available time slot θ_k and securely sends the index θ_k to the server. Fourth, the server (i) replaces all availabilities other than θ_k in E^{θ}_{sched} with random numbers, (ii) reverts the permutation θ, and (iii) sends this new vector(s) $\hat{E}_{sched}$ to each user. Finally, each user decrypts and reverts the initial permutation σ of the received vector(s) and determines which time slot j is the only commonly available time slot.

This simple solution that reveals only a single available time slot to all the participants involves one extra message exchange between the initiator and the scheduling server, as shown in step 3 of Figure 9. Although the permutation θ performed by the server preempts the initiator from knowing the true common availabilities, he might still want to maliciously modify the permuted availabilities. However, the only action the initiator can do is to choose one of the permuted time slots and communicate its index θ_k to the server, as it is the server who will then revert the permutation θ and send the final vector(s) $\hat{E}_{sched}$ to all users.

10. CONCLUSION AND FUTURE WORK

Activity scheduling applications are increasingly used by people on-the-move to efficiently and securely manage their time. In addition to privacy, which is paramount, such services should also be practical and feasible to implement, given the client-server paradigm that most providers are using. In this paper, we have provided a framework for the formal study of privacy properties in such applications, and we have proposed three novel privacy-preserving pro-

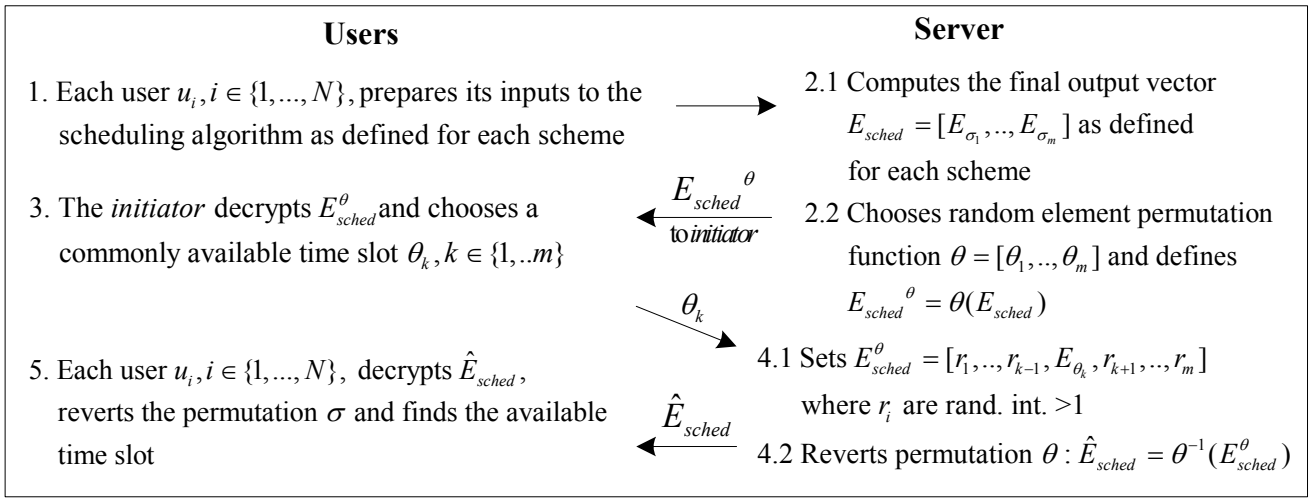


Figure 9: Extended algorithm scheme for revealing a single available time slot.

tocols that, in addition to guaranteeing privacy, are more efficient than similar solutions in terms of computation and communication complexities. Moreover, the implementation and extensive performance evaluation on real mobile devices showed that our privacy-preserving schemes are well suited to practical network architectures and services.

As part of our future work, we intend to further optimize the implementation of the proposed scheduling algorithms for performance on mobile devices, and to include user preferences and security related features described in the previous section. We also plan to release the source code of the proposed scheduling schemes to the general public under the GPL licence.

Acknowledgment

We would like to thank Sudeep Singh Walia and Praveen Kumar for the implementations, Mathias Humbert, Anthony Durussel and Gianpaolo Perrucci for their constructive input that helped improving the quality of this work, as well as the Nokia Research Center (Lausanne) for supporting this project.

11. REFERENCES

[1] Apple iCal. http://apple.com/ical.

[2] Chilabs PDA (Personal Digital Assistants) use study. http://personal.bgsu.edu/~nberg/chilabs/pda.htm.

[3] Doodle: easy scheduling. http://www.doodle.com/.

[4] Google smart rescheduler. http://gmailblog.blogspot.com/2010/03/smart-rescheduler-in-google-calendar.html.

[5] Microsoft Outlook. http://office.microsoft.com/outlook.

[6] Nokia Ovi. http://ovi.nokia.com.

[7] dailywireless.org. http://www.dailywireless.org/2009/03/24/smartphone-users-100m-by-2013, 2009.

[8] I. Bilogrevic, M. Jadliwala, J.-P. Hubaux, I. Aad, and V. Niemi. Privacy-preserving activity scheduling on mobile devices. EPFL Technical Report 161569, https://infoscience.epfl.ch/record/161569, 2010.

[9] C. Cachin and R. Strobl. Asynchronous group key exchange with failures. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 357–366, New York, NY, USA, 2004. ACM.

[10] C.-H. O. Chen, C.-W. Chen, C. Kuo, Y.-H. Lai, J. M. McCune, A. Studer, A. Perrig, B.-Y. Yang, and T.-C. Wu. Gangs: Gather, authenticate 'n group securely. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 92–103, New York, NY, USA, 2008. ACM.

[11] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. *Financial Cryptography and Data Security FC'10*, 2010.

[12] W. Du and M. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22. ACM New York, NY, USA, 2001.

[13] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[14] E. Ephrati, G. Zlotkin, and J. S. Rosenschein. Meet your destiny: A non-manipulable meeting scheduler. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 359–371, New York, NY, USA, 1994. ACM.

[15] M. Franzin, E. Freuder, F. Rossi, and R. Wallace. Multi-agent meeting scheduling with preferences: Efficiency, privacy loss, and solution quality. *Computational Intelligence*, 20(2), 2004.

[16] O. Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[17] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984.

[18] T. Herlea, J. Claessens, B. Preneel, G. Neven, F. Piessens, and B. De Decker. On securely scheduling a meeting. In *Trusted information: the new decade challenge: IFIP TC11 16th International Conference on Information Security (IFIP/Sec'01), June 11-13, 2001, Paris, France*, pages 183–198. Kluwer Academic Pub, 2001.

[19] B. Kellermann and R. Böhme. Privacy-Enhanced Event Scheduling. In *IEEE International Conference on Computational Science and Engineering*, volume 3, pages 52–59, 2009.

[20] L. Kissner and D. Song. Privacy-preserving set operations. *Advances in Cryptology - CRYPTO 2005*, 3621:241–257, 2005.

[21] Y.-H. Lin, A. Studer, H.-C. Hsiao, J. M. McCune, K.-H. Wang, M. Krohn, P.-L. Lin, A. Perrig, H.-M. Sun, and B.-Y. Yang. Spate: Small-group PKI-less authenticated trust establishment. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 1–14, New York, NY, USA, 2009. ACM.

[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology - EUROCRYPT '99*, 1592:223–238, 1999.

[23] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):126, 1978.

[24] M. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.

[25] M. C. Silaghi. Meeting scheduling guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any discsp). In *WI '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 711–715, Washington, DC, USA, 2004. IEEE Computer Society.

[26] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology - EUROCRYPT '96*, pages 190–199, 1996.

[27] R. Wallace and E. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161(1-2):209–227, 2005.

[28] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161(1-2):229 – 245, 2005. Distributed Constraint Satisfaction.

[29] A. Zunino and M. Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 36(3, Part 2):7011 – 7018, 2009.